2.16 NETWORK COMMUNICATIONS

Network communications provide a commander with the capability to transmit intent and orders to subordinates, and they enable subordinate units to send requests, own-force status information, and intelligence data up the various command chain hierarchies. Network communications also connect peers during mission execution. The communications network consists of the communications equipment onboard each platform (see Section 2.7) and each platform's knowledge about how to communicate with other platforms. This knowledge consists of the means of communication (satellite uplink, encrypted teletype, digital data link, two-way landline, or broadcast); the channels, frequencies or data links to use for the various means; and the message types and formats (protocols) for each of the different message types.

Most communication in SWEG requires an explicitly defined network; the only exception is for entities which are modeled as separate components of a single player. These intraplayer components have implicit instantaneous perfect communications. Explicit network communications are completely user defined. The creation of communications networks in SWEG involves three primary steps: network types definition, message protocol definition, and instantiation of a network.

The user defines the types of communications networks that may be used within the scenario. The definition of a network type includes the mode (continuous or intermittent), the time required for all members of the network to change to a new frequency, whether or not to perform an explicit signal calculation, the message types which the net can handle, and the transmit time and priority for each message type that will be transmitted on the network. The typical net types that could be found in a DoD mission-level model might be landline, broadcast and satellite uplink.

The user defines the protocol for each message type that may be used within the scenario. The protocols will include the message's subject, the desired action by the recipient upon message receipt, and the data that comprises the body of the message. SWEG will use the message types in order to create an instance of a message at the time of transmission. Some potential message types include intelligence report, weapons assignment, and cancel assignment.

The user creates an instance of a network type by assigning a unique network identifier to the network type and specifying the network's initial and alternate frequencies. The instantiation is complete when platforms are assigned to the newly created communications network. The assignment of the unique identifier permits SWEG to have multiple instances of the same network type.

Real world entities usually belong to multiple command chain hierarchies, utilize various means of communications, and are members on multiple communications networks. SWEG allows the user to place a platform on any number of nets of any user-defined types; a net can send only one message at a time. Two simultaneous transmissions at the same frequency on different nets will not interfere with each other. The user defines the connectivity between platforms by arbitrarily assigning them to nets; thus any two platforms can communicate. The user also defines tactics and capabilities for the platforms that specify their procedures for transmitting and noticing messages, and for making decisions about taking any actions as a result of receiving a message. Thinking and decision-making are described in Sections 2.9 - 2.11.

2.16.1 Functional Element Design Requirements

This section contains the design requirements necessary to implement the simulation of network communications in SWEG 6.5.

a. SWEG will use networks for communication between players in a scenario and will provide the user with the capability to define the types of networks which may be used.

b. For each net type, SWEG will provide the user with the capability to specify either intermittent or continuous mode for message transmission. Intermittent mode nets are modeled so that emissions from the transmitters occur only when an explicitly modeled message is being sent. Transmitters on continuous mode nets are considered to be continuously transmitting whether or not a specific message is actually being sent. The transmit mode affects the sensing chances for the warning receiver sensors.

c. For each net type, SWEG will provide the user with the capability to define the time delay for changing frequencies; this delay must be greater than or equal to zero. Should the user specify alternate frequencies for any specific net, SWEG will use this time delay when changing to the new frequency. If no alternate frequencies are specified for the net, the time delay is ignored during processing.

d. For each net type, SWEG will provide the user with the capability to specify whether or not signal level is to be explicitly calculated for message transmission. If not, signals will be assumed to be strong enough for all messages to be received on nets of that type. In particular, the net will not be disruptable by jamming.

e. SWEG will provide the user with the capability to specify the protocols or message types that can be transmitted on each type of net. For each protocol, the user can specify a transmit time and a one-way priority. The transmit time represents the elapsed time between transmission and receipt of the message. The priority affects the queuing of messages for the recipient to process (see Sections 2.9 - 2.11 for a discussion of queuing).

f. Within each message type, SWEG will provide the user with the capability to specify an action for the recipient. SWEG will not require that the message recipient act according to the specified action; the response of the recipient will be determined by its decision-making instructions (described in Sections 2.9-2.11). Currently, the user can specify three action categories:

   - Action - This option informs the recipient that a particular action is expected upon receipt.
   - Cancel - This option informs the recipient that a particular pending action should be removed from the pending list.

- Information - This option is used when the purpose of a message is to convey data.

g. SWEG will provide the user with the capability to specify the subject of the message type. Currently, SWEG has seven pre-defined subjects:

- potential target perception
- recipient
- all sensor perceptions of the recipient
- sender
- recipient's subordinates
- recipient's commanders
- any particular player whose relation to the receiver cannot be adequately described by the previously listed options.

h. SWEG will provide the user with the capability to precisely define the data included in a message type. The information included in a message type can be categorized into three classes, and each message type will contain data items for only one class. The first class defines data items related to a specific perception, and any one or more of the following may be included:

- 2D position
- 3D position
- pitch
- attitude
- speed
- local track ID
- global ID
- player type
- platform type
- system status
- communications device frequencies
- sensor frequencies
- sender status
- perceived elements
- perception time
- perception-interaction key
- sender-interaction key

The second class defines the possible modes of control for the recipient to assume, and any one or more of the following may be included:

- assume control of weapon assignments
- assume control of launching
- assume control of jamming

The third class defines a description of some item to be requested of or through the recipient, and any one or more of the following items may be requested:

- a resource
- an amount

- an action
- a quantity
- a destination

i. SWEG will provide the user with the capability to define specific instances of nets by specifying an ID number, the net type, and a frequency, and possible alternate frequencies. Each specific net must have at least one platform assigned to it.

2.16.2 Functional Element Design Approach

Connectivity between players is implemented in the SDB where the net types are defined, and specific communications receivers are assigned to an instance of a net type by specifying an unique net identifier as well as the main and alternate frequencies. Each type of communications network that might be used in the scenario must have a net type entry. This entry defines the network's transmit mode, the time delays in changing frequencies, the requirement of whether or not to calculate a signal level, the types of messages that can be transmitted over the net type, and the time delays and priorities associated with each message type. The message types are defined in the MESSAGE-DEFINITION section of the TDB.

In order for players to communicate with each other, they must have both a communications receiver and transmitter; and the receiver and transmitter must be paired through the linkages data item in the TDB. Even if a player is only going to receive a message, it must have a paired transmitter. The paired receiver and transmitter must belong to the same player, however they can be systems on different elements or different platforms.

Each SDB player assigns communications receivers to a specific net. The communications transmitters are assigned to the net by default since they are paired to a specific receiver. A scenario player becomes a member of a net by specifying the net's identifying number, the net type and a frequency within the system data entry in the SDB's player data item. Alternate frequencies can be defined for a net, and the frequency is changed by implicitly modeled tactics in the physical transmission event.

2.16.3 Functional Element Software Design

This section contains a table and two software code trees which describe the software design necessary to implement the requirements and design approach outlined above. The table lists most of the functions found in the code trees, and a description of each function is provided. Figure 2.16-1 depicts the path from main to yakker, the top-level C++ function within the code for communication events. Figure 2.16-2 contains the code tree for yakker and its subordinate functions.

A function's subtree is provided within the figure only the first time that the function is called. Some functions are extensively called throughout SWEG, and the trees for these functions are in the appendix to this document rather than within each FE description. Within this FE, the functions in that category are MITRcontrol and all member functions in the C++ class WhereIsIt.

Not all functions shown in the figures are included in the table. The omitted entries are trivial lookup functions (single assignment statements), list-processing or memory allocation functions, or C++ class functions for construction, etc.

TABLE 2.26-1. Network Communications Functions.

Function	Description
antgeom	calculates antenna pointing and relative angle data
BaseHost::Run	runs all steps
crslwc	determines line/circle crossing
crslwl	determines line/line crossing
crslwp	determines line/plane crossing
ergatn	calculates attenuation for energy transmission
ergazel	retrieves table entry for gain from azimuth and elevation
erggar	calculates gain and range for energy transmission
featlos	determines if shapes interfere with line of sight
jamcal	calculates jammer interference power at victim receiver
loschk	checks line of site between two objects
main	controls overall execution
MainInit	initiates processing
MainParse	controls parsing of user instructions
numerical	sorts function for address codes
program	controls execution of all steps except bootstrap
redwood	adds new entry to or removes top entry from leftist tree
region	determines if a point is within a two dimensional region
semant	controls semantic processing of instructions
simnxt	controls runtime execution
simphy	controls processing physical events
simul8	controls semantic processing of runtime instructions
srhpro	searches table for an entry containing a specific value
TAddrData::GetJamInteractions	checks for jammer interactions with a communications device
TAddrData::GetParentData	retrieves the TAddrData object from a parent
TAddrData::GetShapeList	finds the shape list at a given address
TAddress::GetAddresses	retrieves a sorted collection of addresses between two points
TAddress::InsertVertCodes	inserts address codes, including parents, for the given point
TMemory::Allocate	allocates permanent storage
TMemory::AllocTemp	allocates temporary storage
TMemory::Deallocate	deallocates storage
TMemory::DeallocFront	deallocates storage
TMemory::LLSTremove	removes a node from a linked list
TMemory::LLSTsearch	searches a list
TMemory::LLSTsearchhard	searches a list using extra parameters

TABLE 2.26-1. Network Communications Functions. (Contd.)

Function	Description
TTerrain::EdgeMasklos	determines the masking of terrain edges
TTerrain::Elevation	determines the z-coordinate on a surface given the x and y coordinates
TTerrain::FindTriangle	determines the terrain triangle for a point given an x,y coordinate pair
TTerrain::LineOfSight	determines if there is a line of sight between two objects
WhereIsIt::CalcPosition	determines position for a platform given a time
WhereIsIt::CalcUnitVel	determines unit velocity for a platform given a time
WhereIsIt::CalcUpVector	determines local up vector given a time
yaeail	adds yet another entry to the scenario action item list
yakker	controls processing of communications events
yaknex	determines next communications event for a given net
yaksig	determines signal level at receiver

```
main
    |-BaseHost::Run
        |-MainInit
            |-program
                |-MainParse
                    |-semant
                        |-simul8
                            |-simnxt
                                |-simphy
                                    |-yakker
                                        |-yaksig
                                        |-yaknex
```

FIGURE 2.16-1. Network Communications Top-Level Code Tree.

```
yakker
      |-TMemory::Index2Ptr
      |-TTable::SearchIGetDouble
      |    \-TTable::RequiredInt
      |         |-TTable::SearchInteger
      |         |    \-TMemory::Ptr2Index
      |         \-TMessages::WriteMessage
      |              |-TMessages::GetMsg
      |              |-TMessages::PrintALine
      |              |    \-TSeqFile::Write
      |              |         \-MFiles::Append
      |              \-sysdun
      |                   |-TActWindow::GetNext
      |                   |-TActWindow::~TActWindow
      |                   \-ProgramStop::ProgramStop
      |                        \-SwegExcpt::SwegExcpt
      |                             \-StringDup
      |-TMaster::GetPlayer
      |    \-TAccDirect::GetItem
      |         \-TAccDirect::GetItem
      |-TPlayer::IsAlive
      |-TMemory::LLSTsearch
      |    |-TMemory::LLSTsearch
      |    |    \-TMemory::Index2Ptr
      |    \-TMemory::Index2Ptr
      |-TMemory::Ptr2Index
      |-yaksig
      |    |-TMemory::Ptr2Index
      |    |-TMemory::LLSTsearch
      |    |-TMemory::LLSTsearchhard
      |    |    |-TMemory::LLSTsearchhard
      |    |    |    \-TMemory::Index2Ptr
      |    |    \-TMemory::Index2Ptr
      |    |-TMemory::Allocate
      |    |    \-TMemory::DoAllocate
      |    |         |-TMemory::GetBlockLength
      |    |         |-CountMemOpns
      |    |         |    \-TMaster::DebugOn
      |    |         |-TMemory::Ptr2Index
      |    |         |-ProgramStop::ProgramStop
      |    |         \-TMemory::WriteSummary
      |    |              |-TMemory::WordsUsed
      |    |              |-TMemory::CalcWdsLeft
      |    |              \-CountMemOpns
      |    |-isZeroEquiv
      |    |-TMemory::Index2Ptr
      |    |-WhereIsIt::CalcPosition
```

FIGURE 2.16-2. Network Communications Code Tree.

```
|    |-loschk
|    |    |-TMaster::GetTerrain
|    |    |-TMaster::TerrainOn
|    |    |-DVector::Getz
|    |    |-dbg_acos
|    |    |    |-MathExcpt::MathExcpt
|    |    |    \-acos_c
|    |    |-operator-
|    |    |-DVector::GetHorizLength
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    \-TTerrain::LineOfSight
|    |         |-DVector::Getz
|    |         |-DVector::Getx
|    |         |-DVector::Gety
|    |         |-dist
|    |         |-dbg_acos
|    |         |-VertexIndex::VertexIndex
|    |         |    \-VertexIndex::operator=
|    |         |-TTerrain::FindTriangle
|    |         |    |-VertexIndex::VertexIndex
|    |         |    |-TAddress::Cartesian2Spherical
|    |         |    |    |-dbg_sqrt
|    |         |    |    |-dbg_asin
|    |         |    |    |    |-MathExcpt::MathExcpt
|    |         |    |    |    \-asin_c
|    |         |    |    \-dbg_atan2
|    |         |    |         \-MathExcpt::MathExcpt
|    |         |    |              \-SwegExcpt::SwegExcpt
|    |         |    |-TMessages::WriteMessage
|    |         |    |-TAddress::Spherical2Cartesian
|    |         |    |    \-TMaster::DebugOn
|    |         |    |-VertexIndex::operator=
|    |         |    |-VerticeArray::operator[]
|    |         |    |    \-VertexIndex::Value
|    |         |    |-TTerrain::toPtr
|    |         |    |    \-SwegExcpt::SwegExcpt
|    |         |    |-operator-
|    |         |    |    \-VertexIndex::VertexIndex
|    |         |    |-VertexIndex::operator++
|    |         |    |-dist
|    |         |    |-operator+
|    |         |    |    \-VertexIndex::VertexIndex
|    |         |    \-VertexIndex::operator+=
|    |         |         \-operator+
|    |         \-TTerrain::EdgeMasklos
|    |              |-dist
|    |              |-VerticeArray::operator[]
```

FIGURE 2.16-2. Network Communications Code Tree. (Contd.)

```
|    |              |-operator+
|    |              |-VertexIndex::operator+=
|    |              |-isZeroEquiv
|    |              \-TTerrain::toIndex
|    |                   \-SwegExcpt::SwegExcpt
|    |-featlos
|    |    |-TMaster::GetTerrain
|    |    |-TMaster::TerrainOn
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    |-DVector::Getz
|    |    |-TTerrain::Elevation
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Gety
|    |    |    |-VertexIndex::VertexIndex
|    |    |    |-TTerrain::FindTriangle
|    |    |    |-VerticeArray::operator[]
|    |    |    |-operator+
|    |    |    \-isZeroEquiv
|    |    |-TMemory::Index2Ptr
|    |    |-TMemory::AllocTemp
|    |    |    \-TMemory::DoAllocate
|    |    |-sorted_collection::sorted_collection
|    |    |-TAddress::GetAddresses
|    |    |    |-TAddress::GetCode
|    |    |    |    |-DVector::Getx
|    |    |    |    |-DVector::Gety
|    |    |    |    \-TMessages::WriteMessage
|    |    |    |-TAddress::InsertVertCodes
|    |    |    |    |-sorted_collection::insert_nodup
|    |    |    |    |    |-MTree::insert_nodup
|    |    |    |    |    |    |-MTree::insert_nodup
|    |    |    |    |    |    \-MTree::MTree
|    |    |    |    |    \-MTree::MTree
|    |    |    |    \-numerical
|    |    |    |-operator-
|    |    |    |-operator^
|    |    |    |-TAddress::GetCellRadius
|    |    |    |-dbg_sqrt
|    |    |    |-operator*
|    |    |    |-DVector::operator+=
|    |    |    \-operator+
|    |    |-sorted_collection::getfirst
|    |    |-sorted_collection::getnext
|    |    |    \-MTree::getnext
|    |    |-TAddress::GetShapeList
|    |    |    |-TAddrNode::DataPresent
|    |    |    |    \-TAddrNode::GetNode
```

FIGURE 2.16-2. Network Communications Code Tree. (Contd.)

```
|    |    |    |         |-TAddrData::GetCode
|    |    |    |         |-TAddrData::TAddrData
|    |    |    |         |-TAddrNode::TAddrNode
|    |    |    |         |    \-MTree::MTree
|    |    |    |         \-TAddrData::PutNodePtr
|    |    |    \-TAddrData::GetShapeList
|    |    |         \-TMemory::Index2Ptr
|    |    |-TMemory::Ptr2Index
|    |    |-TTable::SearchInt
|    |    |    \-TMemory::Ptr2Index
|    |    |-region
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Gety
|    |    |    |-dbg_atan2
|    |    |    \-dist
|    |    |-crslwp
|    |    |    |-DVector::DVector
|    |    |    |-DVector::DVector
|    |    |    |-dbg_sqrt
|    |    |    |-crslwc
|    |    |    |    |-operator-
|    |    |    |    |-DVector::Putz
|    |    |    |    |-operator^
|    |    |    |    |-dbg_sqrt
|    |    |    |    |-operator+
|    |    |    |    \-operator*
|    |    |    |-TMemory::Index2Ptr
|    |    |    |-DVector::operator
|    |    |    |-crslwl
|    |    |    |    |-operator-
|    |    |    |    |-operator*
|    |    |    |    |-DVector::Getz
|    |    |    |    |-operator+
|    |    |    |    \-operator*
|    |    |    |-TMemory::Allocate
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Gety
|    |    |    \-TMemory::Ptr2Index
|    |    |-DVector::DVector
|    |    |-sorted_collection::delete_collection
|    |    |    \-MTree::delete_tree
|    |    |         |-MTree::delete_tree
|    |    |         \-TMemory::Deallocate
|    |    |              |-TMemory::Deallocate
|    |    |              |    |-TMemory::DeallocFront
|    |    |              |    |    \-TMemory::GetBlockLength
|    |    |              |    |-TMemory::Index2Ptr
|    |    |              |    |-CountMemOpns
```

FIGURE 2.16-2. Network Communications Code Tree. (Contd.)

```
|    |    |             |    \-TMemory::RcylBlock
|    |    |             |         |-TMemory::Index2Ptr
|    |    |             |         \-TMemory::Ptr2Index
|    |    |             \-TMemory::Ptr2Index
|    |    \-TMemory::Deallocate
|    |-erggar
|    |    |-antgeom
|    |    |    |-DVector::DVector
|    |    |    |-WhereIsIt::CalcPosition
|    |    |    |-DVector::operator=
|    |    |    |    |-FloatVector::Getx
|    |    |    |    |-FloatVector::Gety
|    |    |    |    \-FloatVector::Getz
|    |    |    |-DVector::DVector
|    |    |    |    |-FloatVector::Getx
|    |    |    |    |-FloatVector::Gety
|    |    |    |    \-FloatVector::Getz
|    |    |    |-operator-
|    |    |    |-DVector::Norm
|    |    |    |-DVector::GetHorizLength
|    |    |    |-DVector::operator
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Getz
|    |    |    |-DVector::Gety
|    |    |    |-WhereIsIt::CalcUnitVel
|    |    |    |-WhereIsIt::CalcUpVector
|    |    |    |-operator*
|    |    |    |-operator+
|    |    |    |-operator*
|    |    |    |-FloatVector::Getx
|    |    |    |-FloatVector::Gety
|    |    |    |-FloatVector::Getz
|    |    |    |-DVector::VecAng
|    |    |    |    |-operator^
|    |    |    |    |-operator*
|    |    |    |    |-DVector::DVector
|    |    |    |    |-DVector::GetLength
|    |    |    |    |-isZeroEquiv
|    |    |    |    \-dbg_atan2
|    |    |    \-DVector::GetLength
|    |    |-TMemory::Index2Ptr
|    |    |-srhpro
|    |    |-TMemory::Ptr2Index
|    |    \-ergazel
|    |         |-TMemory::Index2Ptr
|    |         |-srhpro
|    |         \-TMemory::Ptr2Index
|    |-DVector::Getz
```

FIGURE 2.16-2. Network Communications Code Tree. (Contd.)

```
|     |-operator-
|     |-DVector::GetHorizLength
|     |-ergatn
|     |     \-srhpro
|     \-dbg_pow
|           \-MathExcpt::MathExcpt
|-jamcal
|     |-TMemory::Index2Ptr
|     |-WhereIsIt::CalcPosition
|     |-TAddrData::GetParentData
|     |     \-TAddrNode::GetParentData
|     |-TAddrData::GetJamInteractions
|     |-TTable::SearchInt
|     \-TMemory::Deallocate
|-TMemory::Allocate
|-yaeail
|     |-TMemory::Index2Ptr
|     |-TMaster::GetGameTime
|     |-TMaster::DebugOn
|     |-TMessages::WriteMessage
|     |-TMemory::Allocate
|     |-TMemory::Ptr2Index
|     |-TMaster::PutScenrTree
|     |-redwood
|     |     |-TMemory::Index2Ptr
|     |     \-TMemory::Ptr2Index
|     |-TMaster::GetScenrTree
|     |     \-TMemory::Index2Ptr
|     \-TMaster::GetPhase
|-yaknex
|     |-yaeail
|     |-TMemory::LLSTremove
|     |     |-TMemory::Index2Ptr
|     |     |-TMemory::LLSTsearch
|     |     |     \-TMemory::Index2Ptr
|     |     \-TMemory::Deallocate
|     |           |-TMemory::Ptr2Index
|     |           |-TMemory::DeallocFront
|     |           |-CountMemOpns
|     |           \-TMemory::RcylBlock
|     \-TMemory::Deallocate
\-TMemory::Deallocate
```

FIGURE 2.16-2. Network Communications Code Tree. (Contd.)

2.16.4 Assumptions and Limitations

- Messages of the same priority are transmitted on a FIFO basis.
- Net transmission capabilities are limited only by the user-defined time delays

for message types sent over the net and the fact that only one message at a time is sent.

- Messages are either received in total or not at all.
- Messages are required to include information, not just an action request.
- Intraplayer communications is implicitly modeled and assumed to be instantaneous and perfect.

2.16.5 Known Problems or Anomalies

None.